

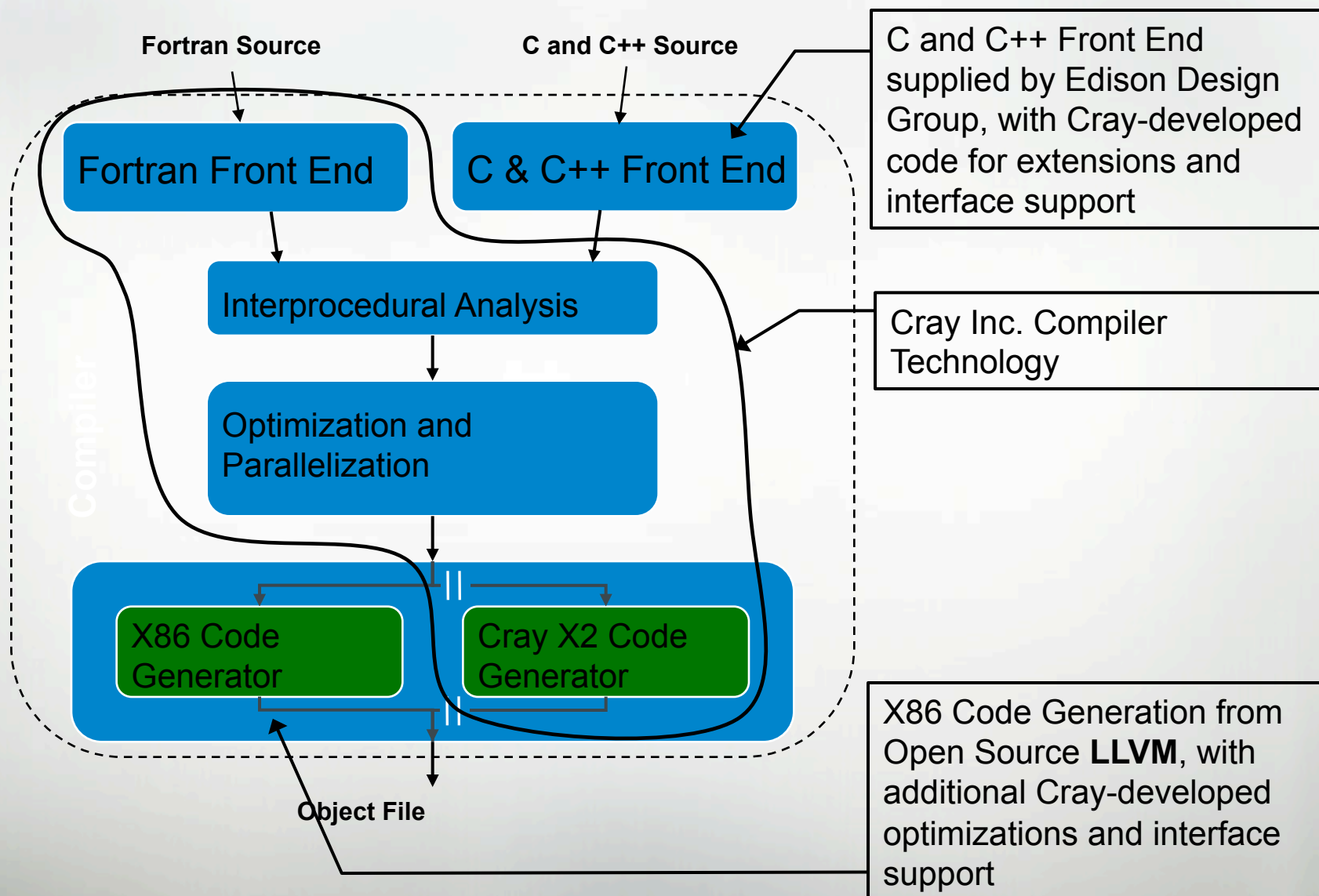
The Cray Compiler Environment

Nathan Wichmann
wichmann@cray.com

Cray Opteron Compiler: Brief History of Time

- Cray has a long tradition of high performance compilers on Cray platforms (Traditional vector, T3E, X1, X2)
 - Vectorization
 - Parallelization
 - Code transformation
 - More...
- Investigated leveraging an open source compiler called LLVM
- First release December 2008

Technology Sources



Why a Cray X86 Compiler?

- Standard conforming languages and programming models
 - Fortran 2003
 - UPC & CoArray Fortran
 - **Fully optimized** and integrated into the compiler
 - No preprocessor involved
 - Target the network appropriately:
 - GASNet with Portals
 - DMAPP with Gemini & Aries
- Ability and motivation to provide high-quality support for custom Cray network hardware
- Cray technology focused on scientific applications
 - Takes advantage of Cray's extensive knowledge of **automatic vectorization**
 - Takes advantage of Cray's extensive knowledge of **automatic shared memory parallelization**
 - **Supplements**, rather than replaces, the available compiler choices

Cray Opteron Compiler: How to use it

- Make sure it is available
 - module avail PrgEnv-cray
- To access the Cray compiler
 - module load PrgEnv-cray
- To target the various chip
 - module load xtpe-[barcelona,shanghi,istanbul]
- Once you have loaded the module “cc” and “ftn” are the Cray compilers
 - **Recommend just using default options**
 - Use `-rm` (fortran) and `-hlist=m` (C) to find out what happened
- `man crayftn`

Cray Opteron Compiler: Current Capabilities

- Excellent Vectorization
 - Vectorize more loops than other compilers
- OpenMP 3.0
 - Task and Nesting
- PGAS: Functional UPC and CAF available today
- C++ Support
- Automatic Parallelization
 - Modernized version of Cray X1 streaming capability
 - Interacts with OMP directives
- Cache optimizations
 - Automatic Blocking
 - Automatic Management of what stays in cache
- Prefetching, Interchange, Fusion, and much more...

Cray Opteron Compiler: Current Strengths

- Loop Based Optimizations
 - Vectorization
 - OpenMP
 - Autothreading
 - Interchange
 - Pattern Matching
 - Cache blocking/ non-temporal / prefetching
- Fortran 2003 Standard; working on 2008
- PGAS (UPC and Co-Array Fortran)
 - Some performance optimizations available in 7.1
- Optimization Feedback: Loopmark
- Focus

Cray Opteron Compiler: Directives

- Cray compiler supports a full and growing set of directives and pragmas

`!dir$ concurrent`

`!dir$ ivdep`

`!dir$ interchange`

`!dir$ unroll`

`!dir$ loop_info [max_trips] [cache_na] ...` Many more

`!dir$ blockable`

man directives
man loop_info

Loopmark: Compiler Feedback

- Compiler can generate an **filename.lst** file.
 - Contains annotated listing of your source code with letter indicating important optimizations

%%% Loopmark Legend %%%

Primary Loop Type

Modifiers

A - Pattern matched

b - blocked

C - Collapsed

f - fused

D - Deleted

i - interchanged

E - Cloned

m - streamed but not partitioned

I - Inlined

p - conditional, partial and/or computed

M - Multithreaded

r - unrolled

P - Parallel/Tasked

s - shortloop

V - Vectorized

t - array syntax temp used

W - Unwound

w - unwound

Example: Cray loopmark messages for Resid

- `ftn -rm ...` or `cc -hlist=m ...`

```
29. b-----<      do i3=2,n3-1
30. b b-----<      do i2=2,n2-1
31. b b Vr--<        do i1=1,n1
32. b b Vr           u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
33. b b Vr           >           + u(i1,i2,i3-1) + u(i1,i2,i3+1)
34. b b Vr           u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
35. b b Vr           >           + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
36. b b Vr-->        enddo
37. b b Vr--<        do i1=2,n1-1
38. b b Vr           r(i1,i2,i3) = v(i1,i2,i3)
39. b b Vr           >           - a(0) * u(i1,i2,i3)
40. b b Vr           >           - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
41. b b Vr           >           - a(3) * ( u2(i1-1) + u2(i1+1) )
42. b b Vr-->        enddo
43. b b----->      enddo
44. b----->        enddo
```

Example: Cray loopmark messages for Resid (cont)

ftn-6289 ftn: VECTOR File = resid.f, Line = 29

*A loop starting at **line 29 was not vectorized** because a recurrence was found on "U1" between lines 32 and 38.*

ftn-6049 ftn: SCALAR File = resid.f, Line = 29

*A loop starting **at line 29 was blocked with block size 4.***

ftn-6289 ftn: VECTOR File = resid.f, Line = 30

A loop starting at line 30 was not vectorized because a recurrence was found on "U1" between lines 32 and 38.

ftn-6049 ftn: SCALAR File = resid.f, Line = 30

A loop starting at line 30 was blocked with block size 4.

ftn-6005 ftn: SCALAR File = resid.f, Line = 31

*A loop starting at **line 31 was unrolled 4 times.***

ftn-6204 ftn: VECTOR File = resid.f, Line = 31

*A loop starting at **line 31 was vectorized.***

ftn-6005 ftn: SCALAR File = resid.f, Line = 37

A loop starting at line 37 was unrolled 4 times.

ftn-6204 ftn: VECTOR File = resid.f, Line = 37

*A loop starting at **line 37 was vectorized.***

Cray Opteron Compiler: Current Weaknesses



- Tuned Performance
 - Vectorization (We vectorize too much)
 - Non-temporal caching
 - Cache blocking
 - Many end-cases
- Spilling
- Scheduling
- Still a young compiler

Byte Swapping

- -hbyteswapio
 - Link time option
 - Applies to all unformatted fortran IO
- Assign command
 - With the PrgEnv-cray module loaded do this:
setenv FILENV assign.txt
assign -N swap_endian g:su
assign -N swap_endian g:du
- Can use assign to be more precise

OpenMP

- OpenMP is **ON** by default
 - Optimizations controlled by `–Othread#`
 - To shut off use `–Othread0` or `–xomp` or `–hnoomp`
- Autothreading is NOT on by default;
 - `-hautothread` to turn on
 - Modernized version of Cray X1 streaming capability
 - Interacts with OMP directives

If you do not want to use OpenMP and have OMP directives in the code, make sure to make a run with OpenMP shut off at compile time

New feature: OMP TASK

- An OpenMP task is an explicit region of code whose execution can be deferred and/or executed in parallel with the surrounding code
 - Completion is guaranteed by synchronization or end of parallel region
 - Must be contained inside a OMP parallel region
 - A task is “put on a queue” to be executed “later”
 - Any thread of the same parallel region that is sitting on a sync point can grab a task off the queue and execute it
- Sort of like “futures” but with limitations
 - Don’t have ID’s, must wait for all or none
 - But maybe are good enough?

Multi-level OpenMP

- Nested OpenMP
 - OMP parallel region inside of an OMP parallel region
 - “New threads” are used at each level
 - Need to use new ENV VARS to control nesting
 - Need to use ENV VARS not in OMP standard for better control
- OMP Tasks inside of parallel regions
 - Can be nested
 - Can be both more and less natural way of programming

Multi-level OpenMP

!\$omp parallel do ...

do i=1,4

call complex_matmul(...)

enddo

Subroutine complex_matmul(...)

!\$omp parallel do private(j,jend,jsize)! num_threads(p2)

do j=1,n,nb

jend = min(n, j+nb-1)

jsize = jend - j + 1

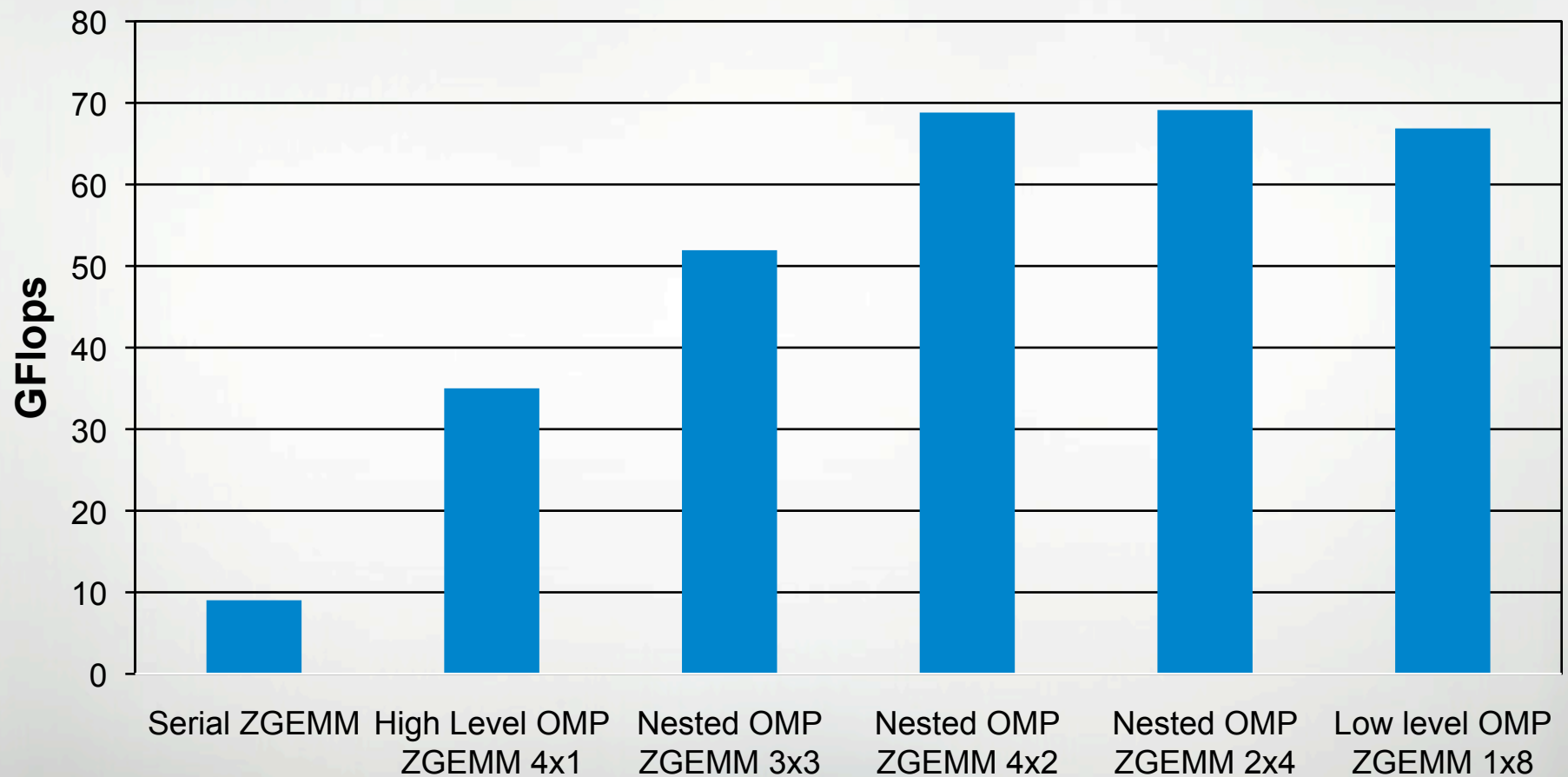
call zgemm(transA,transB, m,jsize,k, &

alpha,A,ldA,B(j,1),ldb, beta,C(1,j),ldC)

enddo

Case Study: PARQUET

4 x ZGEMM 1000x1000

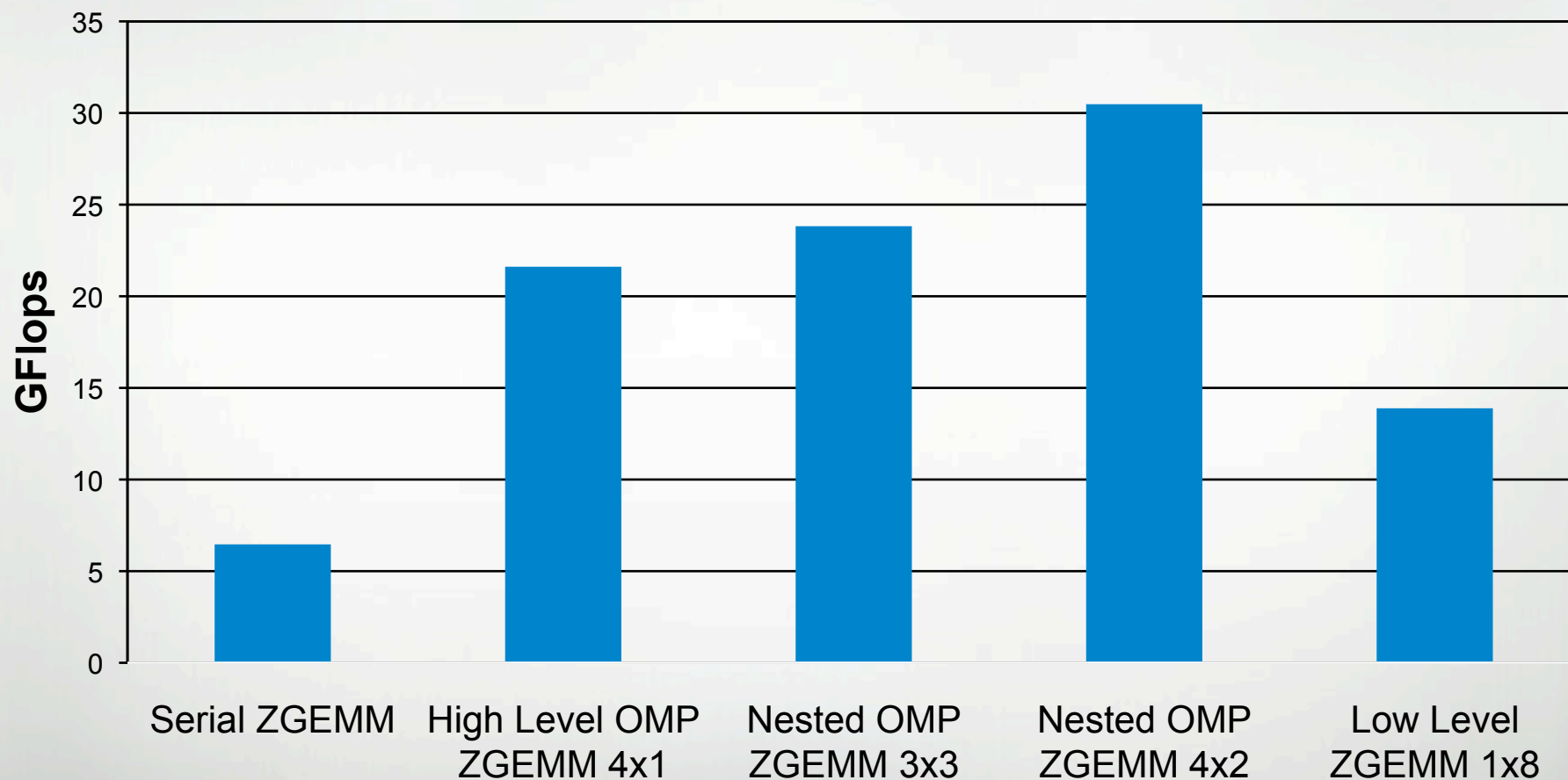


Parallel method and Nthreads at each level

Cray Inc. Confidential

Case Study: PARQUET

4 x ZGEMM 100x100



Parallel method and Nthreads at each level

Lessons from nested parallel regions

- Nested omp can GREATLY expand the amount of parallelism one can attack using OpenMP
- Most people set the environment variable via `omp_num_threads`
 - This, as currently defined, is not adequate for nested parallel regions
 - Using the “`num_threads`” clause may be both tricky and impractical
 - Cray has invented its own `cray_omp_num_threads` variable
- Nested parallel regions is a relatively static distribution
- OMP tasking may be a way of getting around some or all of these issues

Cray Compiler: Future Capabilities

- 7.2 release planned for Q1 10
 - Mostly about performance
 - Magny Cours support

And beyond....

- Fortran 2008
- More tasking capabilities
- Optimized PGAS
- Support for AVX (256 bit vectors)
- Support for Intel

Cray Compiler: Final Thoughts

- Cray Compiler is an interesting alternative for some codes
- Unique and different capabilities can result in significantly different performance.
- Gemini and PGAS will make the Cray compiler even more relevant.

